Synchronous Services

Sally Shlaer
Stephen J. Mellor

1 August 1996

Note to reviewers: This paper is an extract from an early chapter of the upcoming RD book. It should be read **before** *Bridges and Wormholes* (another extract to be released shortly).

Synchronous Services

Sally Shlaer
Stephen J. Mellor

1 August 1996

1. Processing in a Domain

In Shlaer-Mellor models, the processing needed to move objects through their lifecycles is expressed in state models and, at a finer level of detail, in ADFDs. The state models and ADFDs, taken together, provide all of the logic and processing necessary to keep instances of a given domain consistent with one another while accomplishing the mission of the domain.

However, in certain cases, there may be some additional processing that is associated with the domain but is not part of any object's lifecycle. The need for this processing arises because of a requirement to provide information to a neighboring domain. Consider, for example, Figure 1.1. Here the application domain needs to obtain the temperature of a particular magnet. Now let us assume that the Process Input/Output (PIO) domain reads each analog input value periodically from the hardware interface as part of the lifecycle of an Analog Input Point object. The processing required to hand the current value of magnet temperature to the application domain is therefore entirely independent of Analog Input Point's lifecycle: PIO needs simply to pass back the most recent reading of the required point.

Such an operation -- however extensive -- that is not provided in the lifecycle of object(s) in the domain is known as a synchronous service.

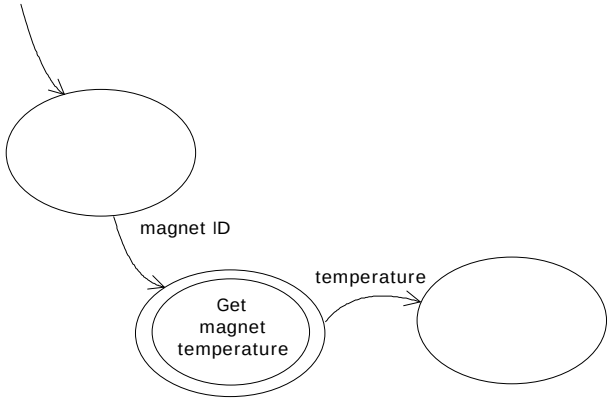

Figure 1.1: The application domain acquires a sensor-based data item from the Process Input/Output domain.

2. Specifying a Synchronous Service

DEFINITION. A *synchronous service* is an operation that is provided by a domain for synchronous[1] invocation by that domain's clients or servers.

[1] "Synchronous" in the analysis sense. See section 3.

The analyst[2] should think of a synchronous service as similar in nature to a function. When the synchronous service is invoked, control is transferred to the service together with values for any input parameters defined by the synchronous service. When the synchronous service is complete, control is returned to the ADFD (or SDFD—soon to be defined) from whence it was called. Values for output parameters, if any, are then made available for use by the caller.

A synchronous service is specified using the same tools we use to specify an action: either by means of a DFD (known as a synchronous service DFD, or SDFD) or in some action specification language. Figure 2.1 shows a schematic rendering of an SDFD.

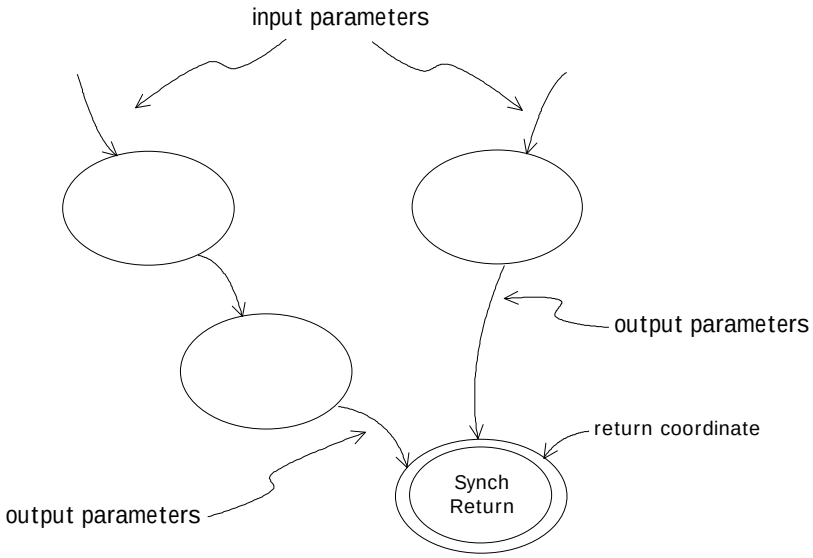

Figure 2.1: An SDFD in schematic form.

The rules for specifying a synchronous service are analogous to those used to specify an action. In particular:

- Each synchronous service must be assigned an identifier unique within the domain. By convention, we name the synchronous services S1, S2, etc. The synchronous service identifier is analogous to the identifier for an action (object.state number).

- All processes on an SDFD must be legal OOA processes: accessors, event generators, tests, transformations, or wormholes.

- Each synchronous service must have a meaning (analogous to the meaning of a state): a name that describes the function carried out by the service.

- A synchronous service may have multiple input parameters. On an SDFD, the input parameters appear like event data items on an ADFD: they are shown on data flows from nowhere.

- A synchronous service always has an input parameter—supplied by the architecture—of type *return coordinate*. This parameter is used to return data to the caller.

---

[2]The architect has a slightly different perspective on how control is passed between the caller and the synchronous service. This is described in *Bridges and Wormholes*.

- A synchronous service may have multiple (or no) output parameters that are to be transmitted to the caller. On an SDFD, the output parameters appear on data flows that are input to a *synchronous return* wormhole: a wormhole whose purpose is to return data to the caller at the point of invocation. Although the return coordinate is not transmitted to the caller, it must also appear as an input to the return wormhole.

- The data type of each input and output parameter must be specified.

- If a synchronous service has no output parameters, no synchronous return wormhole appears on the SDFD because no data is returned to the caller.

These rules are expressed more formally in the extract from the OOA of OOA shown in Figure 2.2.

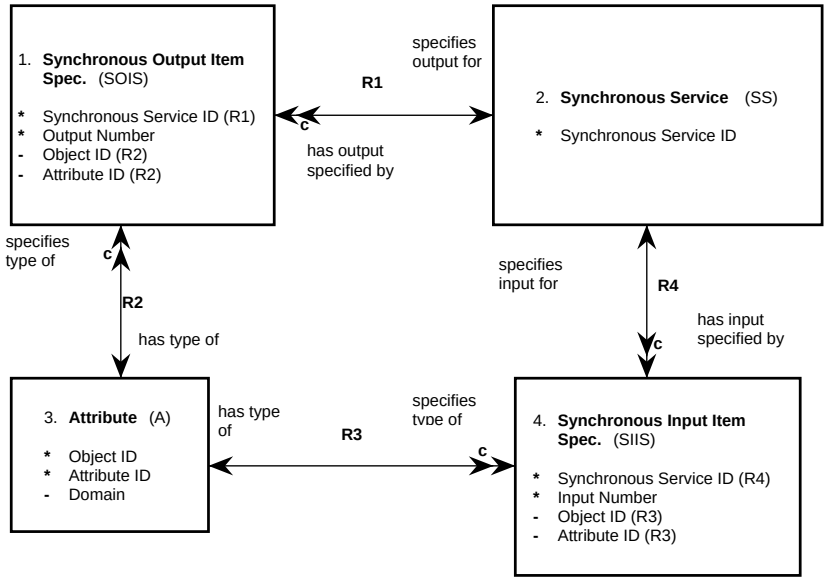

Figure 2.2: Specification of a synchronous service (from the OOA of OOA)

The execution rules for the processes of an SDFD are the same as those for an ADFD:

- A process can execute when all its inputs are available.
- Outputs of a process are available after the process complete executing.
- Input parameters of the SDFD (those shown on data flows from nowhere) are always available.
- Data from data stores is always available.

3. Is a Synchronous Service Really Synchronous?

The Shlaer-Mellor method offers a number of ways of expressing how control is transferred from one unit of processing to the next: For example, control is transferred from one action to another by means of events, and control is given to each process on an ADFD or SDFD -- apparently synchronously -- in accordance with prescribed execution rules. So while we use both synchronous and asynchronous schemes for transfer of control in the analysis models, it is important to remember that what we are actually specifying is the order of execution (or, more precisely, constraints on the order of execution), and not the method of control transfer.

When it comes to implementation, the style of control transfer (synchronous or asynchronous) is prescribed by the architecture. Hence, a control transfer that was modeled as asynchronous may, in fact, be implemented as a synchronous invocation, while a synchronous transfer may be implemented asynchronously.

Acknowledgments. The initial concept of a synchronous service was developed by Ian Wilkie, David Walker, Chris Raistrick, Adrian King, Mike Clarke, and Colin Carter of Kennedy-Carter (U.K.). We are pleased to acknowledge here their contributions to this as well as other aspects of the Shlaer-Mellor method.